# Operating Systems Lab: Introduction to Git

Git is a distributed revision control and source code management system with an emphasis on speed. Git was initially designed and developed by Linus Torvalds for Linux kernel development. Git is a free software distributed under the terms of the GNU General Public License version 2. This tutorial explains how to use Git for project version control in a distributed environment while working on web-based and non web-based applications development.

## Version Control System

Version Control System (VCS) is a software that helps software developers to work together and maintain a complete history of their work.

Listed below are the functions of a VCS −

* Allows developers to work simultaneously.
* Does not allow overwriting each other’s changes.
* Maintains a history of every version.

Following are the types of VCS −

* Centralized version control system (CVCS).
* Distributed/Decentralized version control system (DVCS).

In this chapter, we will concentrate only on distributed version control systems and especially on Git. Git falls under the distributed version control system.

## Distributed Version Control System

Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. But the major drawback of CVCS is its single point of failure, i.e., failure of the central server. Unfortunately, if the central server goes down for an hour, then during that hour, no one can collaborate at all. And even in the worst case, if the disk of the central server gets corrupted and proper backup has not been taken, then you will lose the entire history of the project. Here, the distributed version control system (DVCS) comes into picture.

DVCS clients not only check out the latest snapshot of the directory but they also fully mirror the repository. If the server goes down, then the repository from any client can be copied back to the server to restore it. Every checkout is a full backup of the repository. Git does not rely on the central server and that is why you can perform many operations when you are offline. You can commit changes, create branches, view logs, and perform other operations when you are offline. You require a network connection only to publish your changes and take the latest changes.

### Advantages of Git

**Free and open source**

Git is released under GPL’s open source license. It is available freely over the internet. You can use Git to manage property projects without paying a single penny. As it is an open source, you can download its source code and also perform changes according to your requirements.

**Fast and small**

As most of the operations are performed locally, it gives a huge benefit in terms of speed. Git does not rely on the central server; that is why, there is no need to interact with the remote server for every operation. The core part of Git is written in C, which avoids runtime overheads associated with other high-level languages. Though Git mirrors the entire repository, the size of the data on the client side is small. This illustrates the efficiency of Git at compressing and storing data on the client side.

**Implicit backup**

The chances of losing data are very rare when there are multiple copies of it. Data present on any client side mirrors the repository, hence it can be used in the event of a crash or disk corruption.

**Security**

Git uses a common cryptographic hash function called secure hash function (SHA1), to name and identify objects within its database. Every file and commit is check-summed and retrieved by its checksum at the time of checkout. It implies that, it is impossible to change file, date, and commit message and any other data from the Git database without knowing Git.

**No need of powerful hardware**

In the case of CVCS, the central server needs to be powerful enough to serve requests of the entire team. For smaller teams, it is not an issue, but as the team size grows, the hardware limitations of the server can be a performance bottleneck. In the case of DVCS, developers don’t interact with the server unless they need to push or pull changes. All the heavy lifting happens on the client side, so the server hardware can be very simple indeed.

**Easier branching**

CVCS uses a cheap copy mechanism. If we create a new branch, it will copy all the code to the new branch, so it is time-consuming and not efficient. Also, deletion and merging of branches in CVCS is complicated and time-consuming. But branch management with Git is very simple. It takes only a few seconds to create, delete, and merge branches.

### DVCS Terminologies

**Local Repository**

Every VCS tool provides a private workplace as a working copy. Developers make changes in their private workplace and after committing, these changes become a part of the repository. Git takes it one step further by providing them a private copy of the whole repository. Users can perform many operations with this repository such as add file, remove file, rename file, move file, commit changes, and many more.

**Working Directory and Staging Area or Index**

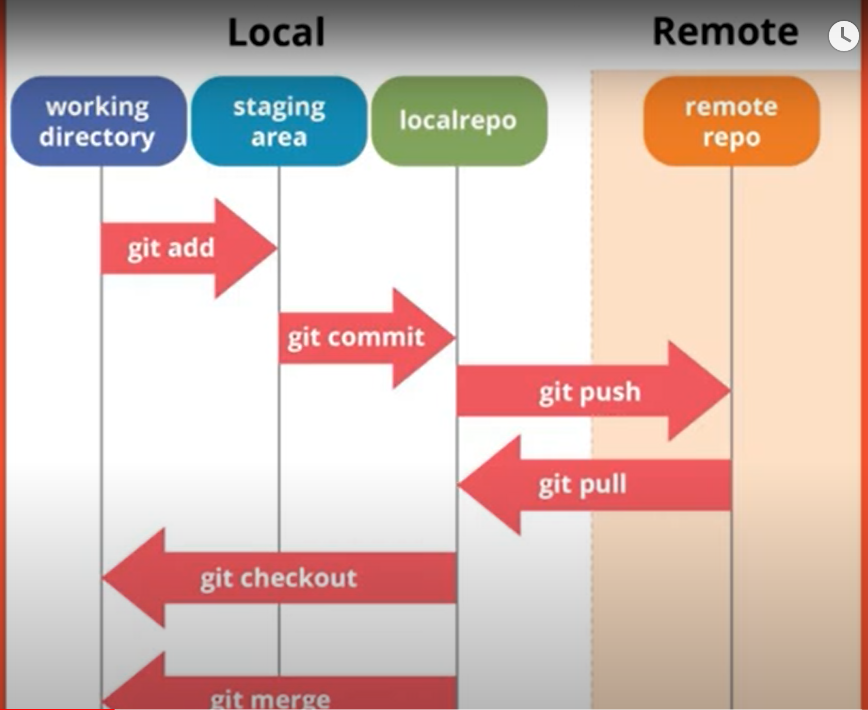
The working directory is the place where files are checked out. In other CVCS, developers generally make modifications and commit their changes directly to the repository. But Git uses a different strategy. Git doesn’t track each and every modified file. Whenever you do commit an operation, Git looks for the files present in the staging area. Only those files present in the staging area are considered for commit and not all the modified files.

Let us see the basic workflow of Git.

Step 1 − You modify a file from the working directory.

Step 2 − You add these files to the staging area.

Step 3 − You perform a commit operation that moves the files from the staging area. After push operation, it stores the changes permanently to the Git repository.



## How to Set Up Git

1. Run the following commands to install Git.

sudo apt update

sudo apt upgrade

sudo apt install git

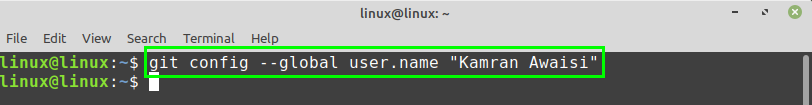
2. Check git version

git –version

3. Configure git on your machine

git config —global user.name <“Your Name”>

[Your name will identify your work]



git config —global user.email <“Your Email”>

4. Confirm your settings.

git config --list

Finally, the git is configured on your system.

## Basic Git Operations

1. Initialize
2. Add
3. Commit
4. Push
5. Pull

## Advanced Git Operations

1. Merging
2. Branching
3. Rebasing

## Create your Own Repository on Git

1. Create a folder using the mkdir command.

mkdir newproject

2. Navigate to the directory using the cd command.

cd newproject

3. Run the following command inside the directory to make it a git repository.

git init

4. Create a sample file.

touch readme.txt

5. Check the status of the git repository.

git status

6. Now, add this file to the Staging Area. Staging Area starts tracking your file and the changes you make in the file

git add readme.txt

Or

git add . (to add all files)

7. Check git status again and notice the difference.

8. Now, commit the file to the repository.

git commit -m “Descriptive Message”

You must describe what you are committing in the descriptive message.

9. Check logs to see commit history

git log

### Create a branch and push it to GitHub

Branches are important, as they allow you to move between project states. Let’s say you want to create a new feature for your game-changing app. To do that, create a new branch. Once you’ve completed work on the feature you can merge this feature from the branch to the master branch. To create the new branch, issue the command:

git checkout -b BRANCH

where BRANCH is the name of the new branch. Once the command completes, issue the command git branch to see that it has been created.

### Push Your Repository on GitHub

1. Create your GitHub account.

2. Login and create a new repository.

3. Now, in the command line, type the following command to connect to the global repository.

git remote add origin URL

git push -u origin master

You will be asked for username and password for GitHub account. Type that and continue with the push operation.

## Your Task

**Create a GitHub account and create a sample repository. Now, install git on your machine. Create a demo project and push changes on your own repo.**

## Terms you must remember

1. Repository
2. Working Directory
3. File Status
4. Staging Area
5. Working Copy